ARHITECTURA SISTEMELOR DE CALCUL SEMINAR 0x03

NOTIȚE SUPORT SEMINAR

Cristian Rusu

ÎNMULȚIREA NUMERELOR, EX. 1

```
       1 0 1 0      a = 10
       1 1 0 1      b = 13
  -----------------
       1 0 1 0      s = 130
     0 0 0 0
   1 0 1 0
 1 0 1 0
  -----------------
 1 0 0 0 0 0 1 0
```

ÎNMULȚIREA NUMERELOR, EX. 2

1 0 1 0	*a*
1 1 0 1	*b*
0 0 0 0	*s*

ÎNMULȚIREA NUMERELOR, EX. 2

1 1 1 1 1 0 1 0 $a = -6$

1 1 1 1 1 1 0 1 $b = -3$

---

1 1 1 1 1 0 1 0 $s = 18$

0 0 0 0 0 0 0 0

1 1 1 1 1 0 1 0

1 1 1 1 1 0 1 0

⋮

---

0 0 0 1 0 0 1 0

ÎNMULȚIREA RAPIDĂ, EX. 3

- **a x 2**
  - soluția: a << 1, sau a + a
- **a x 16**
  - soluția: a << 4
- **a x 3**
  - soluția: a << 1 + a
- **a x 7**
  - soluția: a << 3 – a
- **a / 8**
  - soluția: a >> 3
- **a mod 16**
  - soluția: a & 0x000F
- **a x 72**
  - soluția: a << 6 + a << 3

ÎNMULȚIREA RAPIDĂ, EX. 3

- **a mod 16**
  - soluția: a & 0x000F

- **a div 16**
  - soluția: (a & FFF0) >> 4, sau doar a >> 4

  - de asemenea: a = a & FFF0 + a & 000F = div cu 16 + mod cu 16

ÎNMULȚIREA RAPIDĂ, EX. 4

- **a x 2**
  - soluția: a = (a < 0) ? -((-a) << 1)) : a << 1

ÎMPĂRȚIREA, EX. 5

- **101010 / 10**

1 0 1 0 1 0	1 0 1 0 1 0	1 0 1 0 1 0	1 0 1 0
1 0	1 0	1 0	1 0
	0	0 1	0 1 0

1 0 1 0	1 0	1 0	1 0 1 0 1 0	*a = 42*
1 0	1 0	1 0	1 0	*b = 2*
0 1 0 1	0 1 0 1 0	0 1 0 1 0 1	0 1 0 1 0 1	*s = 21*

ÎMPĂRȚIREA, EX. 5

- **101010 / 10**

1 0 1 0 1 0	$a = 42$
1 0	$b = 2$
0 1 0 1 0 1	$s = 21$

ÎMPĂRȚIREA, EX. 5

- **111010101 / 11**

$\boxed{1\ 0}\ 1\ 0\ 1\ 0\ 1$

1 1

---

0 1 0 0

ÎMPĂRȚIREA, EX. 5

- **111010101 / 11**

1 1 1 0 1 0 1 0 1

1 1

---

1 1 1 0 1 0 1 0 1

1 1

0

---

1 1 1 0 1 0 1 0 1

1 1

0 1

---

1 0 1 0 1 0 1

1 1

0 1 0

---

1 0 1 0 1 0 1

1 1

0 1 0 0

---

1 0 1 0 1 0 1

1 1

0 1 0 0 1

---

1 0 0 1 0 1

1 1

0 1 0 0 1 1

---

1 1 0 1

1 1

0 1 0 0 1 1 1

---

0 1

1 1

0 1 0 0 1 1 1 0

---

0 1

acesta este restul, 1 în zecimal

1 1

0 1 0 0 1 1 1 0 0

---

1 1 1 0 1 0 1 0 1

a = 469

1 1

b = 3

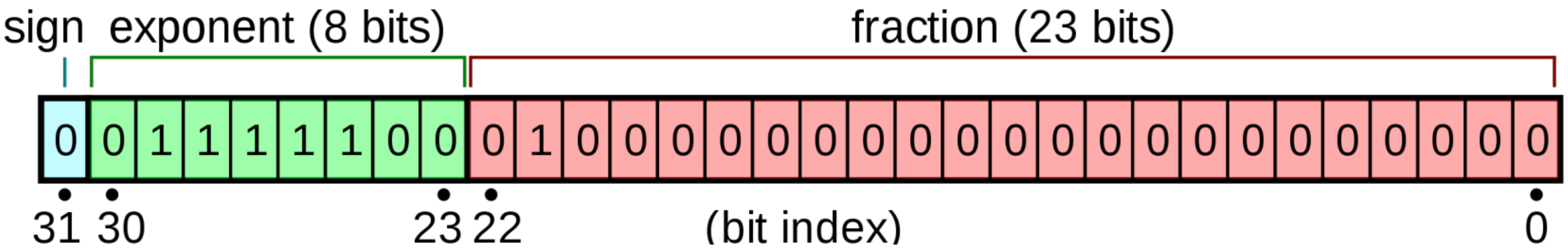
0 1 0 0 1 1 1 0 0

s = 156

0 1

acesta este restul, 1 în zecimal

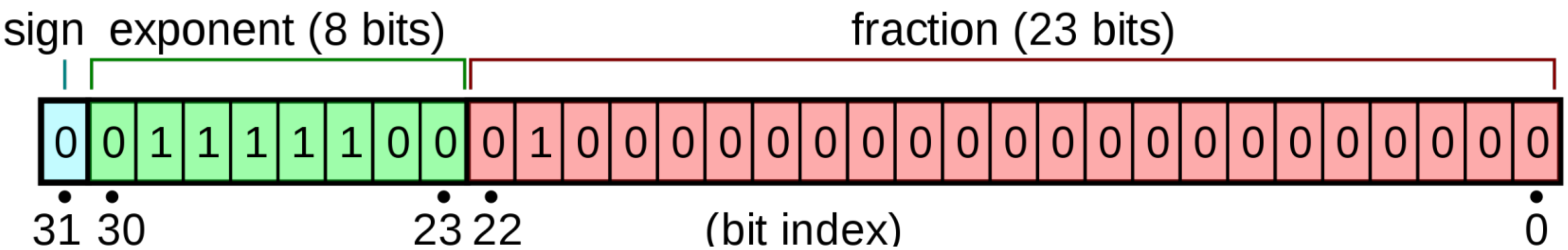
CONVERSIA ÎN IEEE FP, EX. 7



- -1313.3125
  - partea întreagă este: 1313
  - partea fracționară: 0.3125
    - 0.3125 × 2 = 0.625 => 0
    - 0.625 × 2 = 1.25 => 1
    - 0.25 × 2 = 0.5 => 0
    - 0.5 × 2 = 1.0 => 1
  - deci, $1313.3125_{10} = 10100100001.0101_2$
  - normalizare: $10100100001.0101_2 = 1.01001000010101_2 \times 2^{10}$

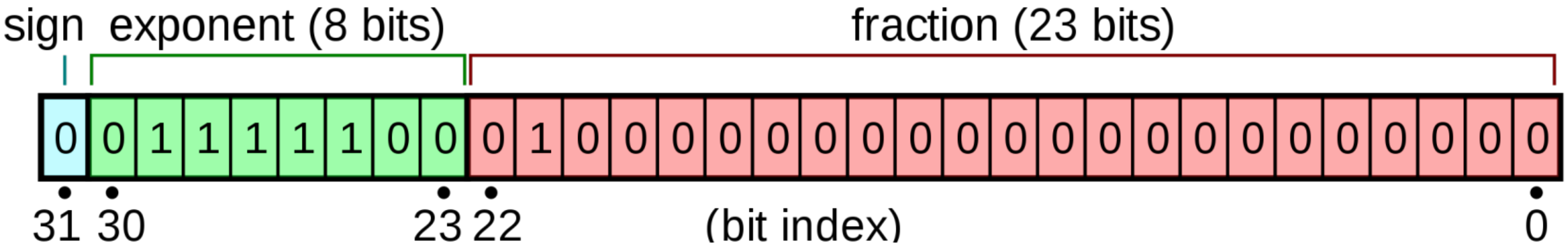
  - mantisa este 01001000010101000000000
  - exponentul este $10 + 127 = 137 = 10001001_2$
  - semnul este 1

CONVERSIA ÎN IEEE FP, EX. 8



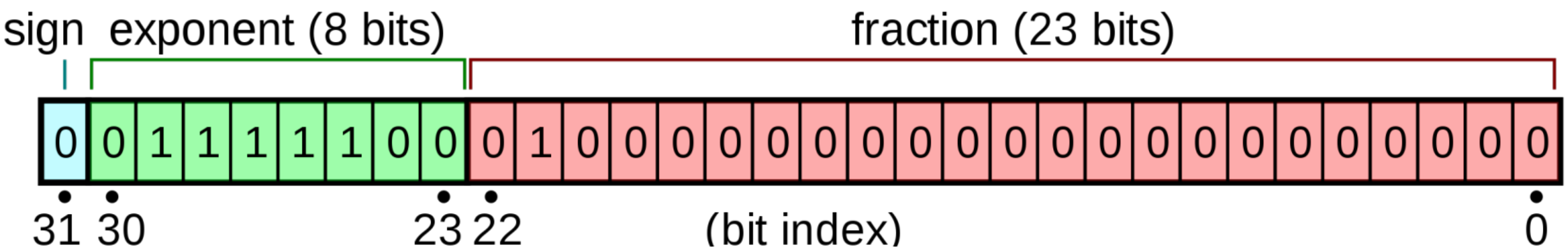
- **schimbați semnul lui a**
  - **soluția: a = a ^ (1 << 31)**
- **calculăm abs(a)**
  - **soluția: a = a & ~(1 << 31)**

CONVERSIA ÎN IEEE FP, EX. 8



- **împărțiți a la 4**
  - **soluția:**
    - **vrem exponentul, unde se află?**
      - **MASK = 0x7F800000**
    - **extragem exponent = (a & MASK) >> 23**
    - **dacă exponent > 1 atunci exponent = exponent – 2, altfel a = 0**
    - **trebuie să actualizăm a**
      - **a = (a & ~MASK) | (exponent << 23)**

CONVERSIA ÎN IEEE FP, EX. 8



- **împărțiți a la 4**
  - **soluția:**
    - **vrem exponentul, unde se află?**
      - **MASK = 0x7F800000**

MASK	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

    - **extragem exponent_var = (a & MASK) >> 23 = 124**

a & MASK	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(a & MASK) >> 23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0

    - **dacă exponent_var > 1 atunci exponent_var = exponent_var – 2, altfel a = 0**

exponent_var – 2 = 122	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0

    - **trebuie să actualizăm a**
      - **a = (a & ~MASK) | (exponent_var << 23)**

a & ~MASK	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exponent_var << 23	0	0	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(a & ~MASK) | (exponent_var << 23)	0	0	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FP ÎN HEX, EX. 9

d) 0xDEADBEEF = 0b11011110101011011011111011101111

- S = 1
- E = 10111101
- M = 01011011011111011101111
- (-1)^S 1.M * 2^{E-127} = (-1) 1.01011011011111011101111 $2^{189-127}$

  = - 1.01011011011111011101111 2^{62}

  = -6259853398707798000

e) 0x44361000 = 0b01000100001101100001000000000000

- S = 0
- E = 10001000
- M = 01101100001000000000000
- (-1)^S 1.M * 2^{E-127} = (1) 1. 01101100001000000000000 $2^{136-127}$

  = 1. 01101100001000000000000 2^{9}

  = 728.25

j) 0xC00010FF = 0b11000000000000000001000011111111

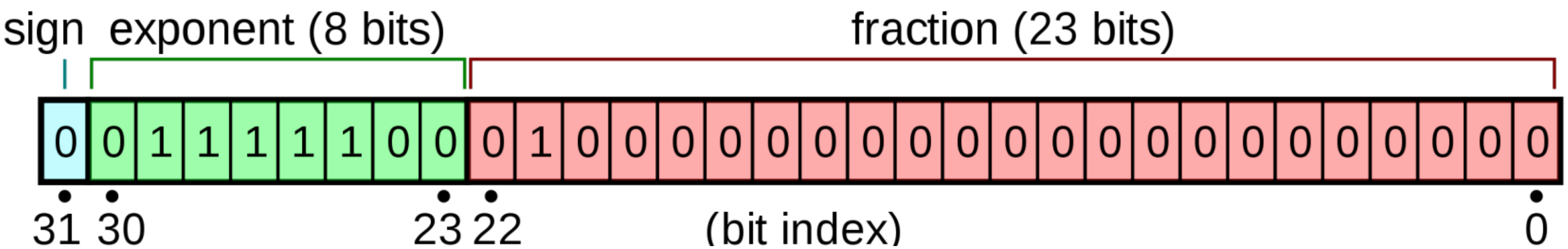
- S = 1
- E = 10000000
- M = 00000000001000011111111
- (-1)^S 1.M * 2^{E-127} = (-1) 1.00000000001000011111111 $2^{128-127}$

  = -1.00000000001000011111111 2^{1}

  = -2.001037359237671

0xFFFFFFFF și 0xFFFF0000 sunt NaN (convenție),

ZERO ÎN IEEE FP, EX. 10



- **setați s = 0, e = 0, f = 0**
- **$a = (-1)^0 \times 1.00\ldots00 \times 2^{-127} = 2^{-127} \neq 0$**

ADUNARE IEEE 754 FP, EX. 11

- **0.2 + 0.3**

- **primul pas, trecem fiecare număr în format**
  - 0.2 = + 1.10011001100110011001100 x 2^{-3} = 0.199999988079071
  - 0.3 = + 1.00110011001100110011001 x 2^{-2} = 0.29999998211860657

- **al doilea pas, alinierea**
  - 0.2 = + 0.11001100110011001100110|000 x 2^{-2}
  - 0.3 = + 1.00110011001100110011001|000 x 2^{-2}

- **al treilea pas, adunăm**
  - 0.2 + 0.3 = 1.11111111111111111111111|000 x 2^{-2}

- **al patrulea pas, normalizare (dacă e necesar)**
  - 0.2 + 0.3 = 1.11111111111111111111111|00 x 2^{-2}

- **pasul final: + 1.11111111111111111111111 x 2^{-2} = 0.4999999701976776**

ÎMPĂRȚIREA RAPIDĂ, EX. 12

- **a / 19**

$$a \times \frac{1}{19} \approx \frac{a \times \frac{2938661835}{2^{32}} + \frac{a - a \times \frac{2938661835}{2^{32}}}{2^1}}{2^4}$$

$$a \times \frac{1}{19} \approx (a \times 2938661835 \times 2^{-32} + (a - a \times 2938661835 \times 2^{-32}) \times 2^{-1}) \times 2^{-4}$$

$$a \times \frac{1}{19} \approx a \times \frac{7233629131}{137438953472}$$

- **1/19 = 0.05263157894**
- **7233629131 / 137438953472 = 0.05263157895**

.

ÎMPĂRȚIREA RAPIDĂ, EX. 12

- **a / 19**

$$a \times \frac{1}{19} \approx \frac{a \times \frac{2938661835}{2^{32}} + \frac{a - a \times \frac{2938661835}{2^{32}}}{2^1}}{2^4}$$

$$a \times \frac{1}{19} \approx (a \times 2938661835 \times 2^{-32} + (a - a \times 2938661835 \times 2^{-32}) \times 2^{-1}) \times 2^{-4}$$

$$a \times \frac{1}{19} \approx a \times \frac{7233629131}{137438953472}$$

- **soluția generală**

$$\frac{a}{D} \approx \frac{\frac{aC}{2^X} + \frac{a - \frac{aC}{2^X}}{2^Y}}{2^Z}$$

$$D \approx \frac{2^{X+Y+Z}}{C \times (2^Y - 1) + 2^X}$$

RACHETE RAPIDE, EX. 19

- **calculați aproximarea binară**
    - soluția: 0.099999904632568359375

- **care este diferența dintre valoarea calculată și 0.1**
    - soluția: 0.1 - 0.099999904632568359375

- **care este eroarea (de timp) după 100 de ore de operare**
    - soluția: 100x60x60x10x(0.1 - 0.099999904632568359375) ≈ 0.34

- **care este eroarea dacă reprezentăm 0.1 în formatul IEEE 754 FP?**
    - soluția: 100x60x60x10x(0.1 - 0.0999999940395352) ≈ 0.021

- **dacă rachetele SCUD pot atinge o viteza MACH 5, care este distanța pe care racheta o poate parcurge în timpul eroare calculat?**
    - soluția: 1715 m/s * 0.34 s ≈ 583 m, 1715 m/s * 0.021 s ≈ 36 m

FAST INVERSE SQUARE ROOT, Q III

```
552 float Q_rsqrt( float number )
553 {
554         long i;
555         float x2, y;
556         const float threehalfs = 1.5F;
557 
558         x2 = number * 0.5F;
559         y  = number;
560         i  = * ( long * ) &y;                       // evil floating point bit level hacking
561         i  = 0x5f3759df - ( i >> 1 );               // what the duck?
562         y  = * ( float * ) &i;
563         y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
564 //      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
```

- **prima instrucțiune interesantă e pe linia 560**

- **de exemplu, 0.5 este:**

0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **aceeași biți dar văzuți de data asta ca un întreg:**
  - $2^{24} + 2^{25} + 2^{26} + 2^{27} + 2^{28} + 2^{29} = 1056964608$ (atât este i)
  - de ce avem nevoie de i?

https://github.com/id-Software/Quake-III-Arena/blob/dbe4ddb10315479fc00086f08e25d968b4b43c49/code/game/q_math.c

FAST INVERSE SQUARE ROOT, Q III

```
552 float Q_rsqrt( float number )
553 {
554         long i;
555         float x2, y;
556         const float threehalfs = 1.5F;
557 
558         x2 = number * 0.5F;
559         y  = number;
560         i  = * ( long * ) &y;                       // evil floating point bit level hacking
561         i  = 0x5f3759df - ( i >> 1 );               // what the duck?
562         y  = * ( float * ) &i;
563         y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
564 //      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
```

- **prima instrucțiune interesantă e pe linia 560**

- **dacă avem în M mantisa și exponentul în E atunci**
  - numărul nostru în FP este 2^{23} x E + M
  - iar valoarea numărului este (1 + M / 2^{23}) x 2^{E-127}
  - observăm că: $$\begin{aligned}\log_2\left(\left(1+\frac{M}{2^{23}}\right)\times 2^{E-127}\right) &= \log_2\left(1+\frac{M}{2^{23}}\right)+\log_2\left(2^{E-127}\right)\\ &= \log_2\left(1+\frac{M}{2^{23}}\right)+E-127\\ &\approx \frac{M}{2^{23}}+E-127+\mu \quad \text{(am folosit } \log_2(1+x)\approx x, \mu \text{ este o corectie)}\\ &= \frac{1}{2^{23}}(2^{23}\times E+M)+\mu-127\\ &= \frac{1}{2^{23}}(\text{reprezentarea pe biti})+\mu-127\end{aligned}$$

https://github.com/id-Software/Quake-III-Arena/blob/dbe4ddb10315479fc00086f08e25d968b4b43c49/code/game/q_math.c

FAST INVERSE SQUARE ROOT, Q III

```
552 float Q_rsqrt( float number )
553 {
554         long i;
555         float x2, y;
556         const float threehalfs = 1.5F;
557 
558         x2 = number * 0.5F;
559         y  = number;
560         i  = * ( long * ) &y;                       // evil floating point bit level hacking
561         i  = 0x5f3759df - ( i >> 1 );               // what the duck?
562         y  = * ( float * ) &i;
563         y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
564 //      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
```

- **prima instrucțiune interesantă e pe linia 560**

- **de ce calculăm log(y)? defapt vrem 1/sqrt(y). dar:**

$$\log_2\left(\frac{1}{\sqrt{y}}\right) = \log_2\left(y^{-1/2}\right) = -\frac{1}{2}\log_2(y) = -(i \gg 1)$$

- **suntem pe linia 561 acum. de unde apare acel număr hexa?**

https://github.com/id-Software/Quake-III-Arena/blob/dbe4ddb10315479fc00086f08e25d968b4b43c49/code/game/q_math.c

FAST INVERSE SQUARE ROOT, Q III

```
552  float Q_rsqrt( float number )
553  {
554          long i;
555          float x2, y;
556          const float threehalfs = 1.5F;
557
558          x2 = number * 0.5F;
559          y  = number;
560          i  = * ( long * ) &y;                                      // evil floating point bit level hacking
561          i  = 0x5f3759df - ( i >> 1 );              // what the duck?
562          y  = * ( float * ) &i;
563          y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
564  //      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
```

- **suntem pe linia 561 acum. de unde apare acel număr hexa?**

$$Y = \frac{1}{\sqrt{y}} \text{ atunci } \log_2(Y) = \log_2(\frac{1}{\sqrt{y}}) \text{ rezulta}$$

$$\frac{1}{2^{23}}(M_Y + 2^{23} \times E_Y) + \mu - 127 = -\frac{1}{2}\left(\frac{1}{2^{23}}(M_y + 2^{23} \times E_y) + \mu - 127\right)$$

$$M_Y + 2^{23} \times E_Y = \frac{3}{2} \times 2^{23}(127 - \mu) - \frac{1}{2}(M_y + 2^{23} \times E_y)$$

$$= \text{0x5f3759df} - (i \gg 1) \quad (\mu \text{ este ales pentru a minimiza eroarea})$$

- **iar apoi pe linia 562 y este transformat înapoi în FP**

https://github.com/id-Software/Quake-III-Arena/blob/dbe4ddb10315479fc00086f08e25d968b4b43c49/code/game/q_math.c

FAST INVERSE SQUARE ROOT, Q III

```c
552 float Q_rsqrt( float number )
553 {
554         long i;
555         float x2, y;
556         const float threehalfs = 1.5F;
557
558         x2 = number * 0.5F;
559         y  = number;
560         i  = * ( long * ) &y;                       // evil floating point bit level hacking
561         i  = 0x5f3759df - ( i >> 1 );               // what the duck?
562         y  = * ( float * ) &i;
563         y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
564 //      y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
```

- **pe linia 563 e o iterație din metoda lui Newton**
    - îmbunătățește rezultatul (e o metodă de optimizare)

https://github.com/id-Software/Quake-III-Arena/blob/dbe4ddb10315479fc00086f08e25d968b4b43c49/code/game/q_math.c